MQTT and IBM MessageSight: Secure, reliable communications for the next generation of resilient mobile applications

Nick Maynard

January 28, 2015

MQTT is a publish-subscribe based lightweight message protocol. A critical enabler of the Internet of Things (IoT), MQTT can also be used for reliable enterprise messaging to mobile devices, enabling secure, reliable communications for the next generation of resilient mobile applications. This tutorial explores the requirements of such a use case, showing how MQTT and IBM® MessageSight are uniquely placed to fulfill them.

Introduction

Enterprise mobile applications are expected to run in a wide variety of situations, and must consider the capabilities of their target platforms. When communicating with backend services, they may have to provide solutions for challenges including:

- Confidentiality of information
- Protocol overhead for low bandwidth connections
- Tolerance of unstable network connections
- Low battery overhead
- Multiple client platforms (iOS, Android, and so on)
- Confidential "push" communication, including background notifications.

Additionally, backend services could need to handle the simultaneous connection of millions of clients.

As defined by mqtt.org, "MQTT is a machine-to-machine (M2M)/'Internet of Things' connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium… It is also ideal for mobile applications because of its small size, low power usage, minimised data packets, and efficient distribution of information to one or many receivers."

This tutorial shows how MQTT and IBM MessageSight are uniquely placed to provide reliable enterprise messaging to mobile devices. It will also outline the risks of such an approach, and show how careful configuration of IBM MessageSight can enable reliability, security, and client confidentiality for these applications.
This article assumes familiarity with HTTP and MQ technologies.

The case for MQTT compares

Let's begin by evaluating the following protocols for communication with backend services and look at the differentiating criteria:

- HTTP
- IBM MQ
- MQTT (IBM MessageSight)

Table 1. Protocol comparison

<table>
<thead>
<tr>
<th>Criteria</th>
<th>HTTP</th>
<th>MQ</th>
<th>MQTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Low protocol overhead</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Unstable network tolerance</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Low power usage</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Millions of connected clients</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Push communication</td>
<td>YES*</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Client platform variance</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Firewall tolerance</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>

- **Low protocol overhead**
  MQTT is unique in that its per-message header can be as low as 2 bytes. MQ and HTTP both have significantly higher per-message overhead. In the case of HTTP, re-establishing the HTTP connection for every new request message causes a significant overhead. **Permanant connections used by MQ and MQTT significantly mitigate this.**

- **Unstable network tolerance**
  MQTT and MQ have the ability to recover from disconnections, and so on, without further code requirements. However, HTTP is unable to do this natively, requiring client-side retry coding that can result in increased problems with idempotency.

- **Low power usage**
  MQTT is uniquely designed for low power usage. **HTTP was not designed with this in mind and consequently has increased power usage.**

- **Millions of connected clients**
  Maintaining millions of simultaneous connections requires significant effort to support on the HTTP stack. Though it is possible, most commercial offerings are not optimised to handle this number of permanent connections. IBM offers IBM MessageSight, which is a single rackmount server that is tested to handle up to one million concurrently connected devices over MQTT. In contrast, MQ is not designed for large numbers of simultaneous clients.

- **Push notification**
  You need to be able to pass notifications to clients in a timely manner. To do this, either a periodic poll, or push methodology must be adopted; push is the best solution from battery, system load and bandwidth points of view.
Our enterprise might require sensitive information to be sent without third-party intermediaries. This discounts OS-specific solutions (such as Apple iOS, Google Play notifications) as the primary transport mechanism.

HTTP only allows push using an approach known as COMET, using long-held HTTP requests. This approach is expensive from both a client and server point of view. Both MQ and MQTT support push as a basic feature of the technology.

• **Client platform variance**
  Both HTTP and MQTT clients have been implemented on a large variety of platforms. MQTT’s simplicity enables implementation on additional clients with minimal effort.

• **Firewall tolerance**
  Some corporate firewalls limit outbound connectivity to certain defined ports. These are often limited to HTTP (port 80), HTTPS (port 443) and others. HTTP can clearly operate in these circumstances. MQTT can be encapsulated in a WebSockets connection, and appears as an HTTP Upgrade request, allowing operation in these circumstances. MQ does not permit this pattern.

### Using MQTT to do this

Many of our required characteristics, such as unstable network tolerance and low power usage, are inherent qualities of the MQTT protocol. As such, they will not be covered here. Instead, we’ll focus on areas that require configuration, and explain the concepts you need to understand to do this correctly.

We’ll use IBM MessageSight to provide the MQTT broker component that our clients connect to. Available in appliance and cloud-deployable virtual image forms, it can be integrated with our enterprise systems and provides an edge-of-network connectivity layer.

> “IBM MessageSight is capable of supporting one million concurrent sensors or smart devices and can scale up to thirteen million messages per second.”

IBM News Release, 29 Apr 2013

Typically, an IBM MessageSight appliance would provide onward connectivity into the main enterprise JMS/MQ service bus, such as IBM Integration Broker. However, it is possible for enterprise applications to communicate directly with clients through the IBM MessageSight MQTT broker.

You need to consider three primary use cases for your MQTT communications: the request-response invocation pattern, streaming, and notifications.

### Request-response

In most applications, the majority of interactions with enterprise services follow the classic request-response pattern:

1. Clients issue request messages over MQTT. The MessageSight broker then passes them onto the backend service.
2. Response messages are passed through the reverse process, from the service via the MessageSight broker, then to the client via MQTT.

Client applications typically use short-term subscriptions to do this, with temporary response queues. A unique response topic for every request-response interaction must be specified in the request message. Typically this is a sub-topic of a designated "user-specific" response topicspace.

Streaming

Typical examples of streamed information will be stock ticker information, currency exchange rates, news, and so on. Streamed information will likely be public in nature. Clients will elect (either implicitly or explicitly) to follow certain streams.

This use case requires client applications to maintain medium-term subscriptions to topics; typically these are active only when the client application's interface is in the "foreground."

Notifications

Notifications can either be user confidential (such as account changes of medical results) or public (such as scheduled downtime or market condition notifications). This use case requires client applications to maintain long-term subscriptions to topics. Additionally, it is expected that the client application will be in a background state, perhaps with the client's device's screen turned off.

Though some mobile operating systems permit permanent connections to MQTT, many do not. In these cases, or in the case where the background connection fails for any reason, a specialised approach is required.

Since v1.1, IBM MessageSight provides **disconnected client notifications** – notifying the client application (via OS-specific push notification services, such as Apple's iOS notification service) that messages are waiting for the client. On receipt of such a notification, the client application should re-connect to the MQTT broker to receive these messages.

Implementation of a disconnected-client aware solution requires development of a separate service that implements IBM MessageSight's disconnected client notification API.

Cautions

Any enterprise MQTT solution will need to carefully consider configuration to allow for the non-functional requirements of security (including client confidentiality) and performance.

Securing IBM MessageSight

Beyond securing the MessageSight instance's administration interfaces (covered in the IBM MessageSight Knowledge Center), you need to consider the following security aspects:

1. **Transport-level security:** Encryption for all communications.
   In our scenario, all communications with IBM MessageSight over the Internet must be encrypted. In addition to industry-standard SSL capabilities, IBM MessageSight provides FIPS 140-2 and NIST 800-131a level encryption capabilities.
2. **Authentication:** Only authorised clients should be allowed to connect.
Optionally, **client certificate authentication** can be used to add extra security. You can use this to ensure that only approved client applications can connect to your MessageSight broker. User authentication should be mandatory. IBM MessageSight authenticates users by checking their credentials with an enterprise LDAP server.

3. **Authorization:** MQTT’s publish-subscribe model is open by default open. You need to ensure confidentiality of information at various levels; for example, enterprise-level, user-level, and device-level.

You need to lock down your topic tree to ensure that users only have access to the information they should. You do this by configuring IBM MessageSight with a number of messaging policies (Table 2). The example topic structure outlined here accounts for the recommendations made by the Fan in per device request-reply scenario from the IBM MessageSight KnowledgeCenter.

**Table 2. Messaging policies**

<table>
<thead>
<tr>
<th>Topic space (base name)</th>
<th>Description</th>
<th>Client subscription rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_NAME/REQUEST/ ${UserID}/${ClientID}</td>
<td>Request topic space for authenticated users.</td>
<td>Publish ONLY by connections with the correct authenticated User ID and Client ID.</td>
</tr>
<tr>
<td>APPLICATION_NAME/RESPONSE/ ${UserID}</td>
<td>Response topic space for authenticated users.</td>
<td>Subscribe ONLY by connections with the correct authenticated User ID.</td>
</tr>
<tr>
<td>APPLICATION_NAME/NOTIFICATION/ ${UserID}</td>
<td>Notification topic space for user-confidential</td>
<td>Subscribe ONLY by connections with correct authenticated User ID.</td>
</tr>
<tr>
<td>APPLICATION_NAME/NOTIFICATION/ PUBLIC</td>
<td>Notification topic space for public notifications.</td>
<td>Subscribe ONLY by connections with correct authenticated User ID.</td>
</tr>
<tr>
<td>APPLICATION_NAME/STREAM</td>
<td>Public stream topic space for all users.</td>
<td>Subscribe ONLY by any authenticated client.</td>
</tr>
</tbody>
</table>

IBM MessageSight enforces these authorization rules by performing **automatic variable substitution in messaging policies**.

The order of substituted variables in the topic name is important. If used, you should specify the ClientID variable as the **last** part of a topic name for substitution. The Client ID is specified by the client when it connects, and can contain any characters, and consequently could be used by an attacker to gain access to sub-topics. However, your client applications and services must subscribe/publish to the correct topics by manually substituting variables to obtain the correct topic names.

**Performance, or choosing the correct Quality of Service**

Your usage patterns - request-response, streaming, and notifications - have unique requirements for the message Quality of Service (QoS) levels you should use.

**Table 3. Quality of Service levels (from the MQTT specification)**

<table>
<thead>
<tr>
<th>QoS level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoS 0: At most once delivery</td>
<td>The message arrives at the receiver either once or not at all. No response is sent by the receiver and no retry is performed by the sender.</td>
</tr>
</tbody>
</table>
It can be tempting to specify a higher QoS than necessary for your messages/subscriptions. However, as QoS increases, there is an increased resource cost on the MessageSight broker. If you want to maximise performance, you must choose the most appropriate QoS level for each message.

**Table 4. Recommended QoS levels**

<table>
<thead>
<tr>
<th>Message type</th>
<th>QoS level</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request-response - request</td>
<td>2: Exactly once</td>
<td>To maintain idempotency without extra client and server-side processing, you must know that each request is delivered exactly once.</td>
</tr>
<tr>
<td>Request-response - response</td>
<td>1: At least once</td>
<td>We have a unique response topic for each request-response interaction, so duplicate delivery is not an issue. The client can simply and safely discard duplicate responses.</td>
</tr>
<tr>
<td>Streaming</td>
<td>0: At most once</td>
<td>Streamed information is typically transitory, so you can afford to lose messages.</td>
</tr>
<tr>
<td>Notifications</td>
<td>2: Exactly once</td>
<td>Notifications, particularly user-specific, are expected to be delivered exactly once to clients.</td>
</tr>
</tbody>
</table>

The information above is simply a recommendation. Your particular enterprise application might have requirements that would alter these suggestions.

**Conclusion**

This tutorial examined the suitability of MQTT for enterprise communications, including a look at potential risks, and outlined configuration policies that can help mitigate these.
Related topics

- Using MQTT Protocol Advantages Over HTTP in Mobile Application Development
- IBM MessageSight Security
- Client certificate authentication
- Variable substitution in messaging policies
- Power Profiling: HTTPS Long Polling vs. MQTT with SSL, on Android
- A Million-user Comet Application with Mochiweb, Part 1
- Using MQTT in Android mobile applications
- Federal Information Processing Standards (FIPS)
- Scenario 5: Fan-in per device request-reply

© Copyright IBM Corporation 2015
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)